

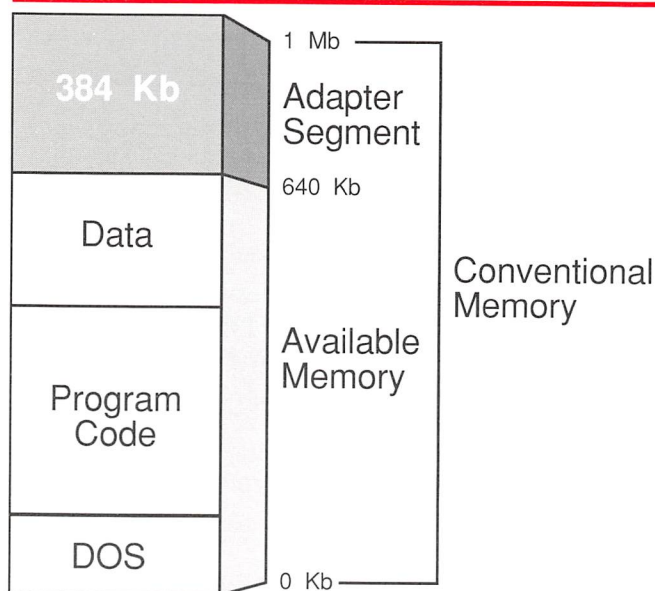
# Inside DOS™

## The memory maze: Sorting out the various types of PC memory

**M**ost DOS users understand that a computer's working memory or RAM (Random Access Memory) is the computer's electronic scratch pad—the place where it loads programs and data. But when it comes to sorting out the different kinds of RAM present in today's computers, many people get lost in the maze of conventional memory, extended memory, and expanded memory. To make matters worse, much of the computer jargon we use to discuss memory makes up an alphabet soup of acronyms (RAM, ROM, LIM, EMS 3.2, EMS 4.0, XMS, etc.).

In this article, we'll help you understand the difference between conventional, expanded, and extended memory. Additionally, we'll explain the difference between real mode and protected mode on the 80286, 80386, and 80486 processors. Finally, we'll help you decide what type of memory you should buy in order to expand the capabilities of your applications.

**Figure A**



The 1 Mb of conventional memory is divided into the adapter segment and the 640 Kb of RAM available to DOS and your applications.

### Conventional memory

Conventional memory is the first 640 kilobytes (Kb) of RAM installed in a personal computer. The first generation of PCs—those based on the 8088 processor, like the IBM PC and PC-XT—are capable of addressing a total of 1 megabyte (1024 Kb) of memory. As you can see in Figure A, however, 384 Kb of that memory—called the *adapter segment*—is reserved for various hardware and system devices, leaving a total of 640 Kb of usable RAM for DOS and applications that run under DOS.

Originally, conventional memory was simply called *memory*; the term *conventional memory* was coined to differentiate it from the other forms of memory that came along later. Technically, the entire first 1 Mb of RAM is conventional memory, but most people use the term to refer to the 640 Kb of RAM available to applications.

A few short years ago, 640 Kb of memory seemed like more than anyone would ever need in a "personal" computer. Computers were sold with a standard amount of 256 Kb, and most software ran with room to spare. However, as software grew more complex, and users became more sophisticated and demanding, the "huge"

*Continued on page 2*

### IN THIS ISSUE

- The memory maze: Sorting out the various types of PC memory ..... 1
- Issue a batch file alert by ringing your computer's bell ..... 5
- Guidelines for managing your fixed disk ..... 6
- Formatting bootable disks from drives D, E, and F ..... 8
- Backing up with XCOPY, revisited ..... 8
- Using the FOR command to issue several commands at once ..... 9
- Tightening up batch files using the FOR command ..... 12



# Inside DOS™

*Inside DOS* (ISSN 1049-5320) is published monthly by The Cobb Group, Inc., 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220. Domestic subscriptions: 12 issues, \$39. Foreign: 12 issues, \$59. Single issues: \$4 domestic, \$6.50 foreign. Send subscriptions, fulfillment questions, and requests for bulk orders to Customer Relations, 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220, or call (800)223-8720 or (502) 491-1900. Our FAX number is 491-4200. Address correspondence and special requests to The Editor, *Inside DOS*, at the address above.

POSTMASTER: Send address changes to The Cobb Group, Inc., P.O. Box 35160, Louisville, KY 40232. Second class postage is paid in Louisville, KY.

Copyright © 1990, The Cobb Group, Inc. All rights reserved. *Inside DOS* is an independently produced publication of The Cobb Group, Inc. No part of this journal may be used or reproduced in any fashion (except in brief quotations used in critical articles and reviews) without the prior consent of The Cobb Group, Inc.

Editor-in-Chief: Mark W. Crane  
Technical Consultant: Tim Landgrave  
Contributing Editor: Van Wolverton  
Editing: Jim Welp  
Toni Frank Bowers  
Production: Eric Paul  
Design: Karl Feige  
Publications Director: Katherine Chesky  
Publisher: Douglas Cobb

The Cobb Group, its logo, and the Satisfaction Guaranteed statement and seal are trademarks of The Cobb Group, Inc. *Inside DOS* is a trademark of The Cobb Group, Inc. Microsoft is a registered trademark of Microsoft Corporation. IBM is a registered trademark of International Business Machines, Inc.

## Conventions

To avoid confusion, we would like to explain a few of the conventions used in *Inside DOS*.

As you probably know, Microsoft has released several versions of DOS. Although most of the articles in this journal apply to all versions of DOS since 2.00, we'll sometimes point out a particular command or technique that requires a particular version. For simplicity, we'll refer to all 4.xx versions as version 4.

Typically, we'll use DOS's default prompt, C>, to represent the DOS prompt in our examples. If you have defined a custom DOS prompt, just assume that the C> represents your prompt.

When we instruct you to type something, those characters usually appear on a separate line along with the DOS prompt. The characters you type will appear in color, while the characters DOS displays will appear in black.

Occasionally, we won't display the command you type on a separate line. In these cases, we'll display the characters you type in italics. For example, we might say, "issue the command *dir \*.txt* at the DOS prompt." Although DOS is not sensitive to capitalization, we'll always display the characters you type in lowercase.

When we refer to a general DOS command (not the command you actually type at the DOS prompt), we'll display that command name in all caps. For example, we might say, "You can use either the COPY or XCOPY command to transfer files from one disk to another."

Many commands accept parameters that specify a particular file, disk drive, or other option. When we show you the form of such a command, its parameters will appear in italics. For example, the form of the COPY command is:

```
copy file1 file2
```

where *file1* and *file2* represent the names of the source file and the target file, respectively.

The names of keys, such as [Shift], [Ctrl], and [F1], appear in brackets. The [Enter] key is represented by the symbol ↵. When two keys must be pressed simultaneously, those key names appear side by side, as in [Ctrl][Break] or [Ctrl]z.

## The memory maze

*Continued from page 1*

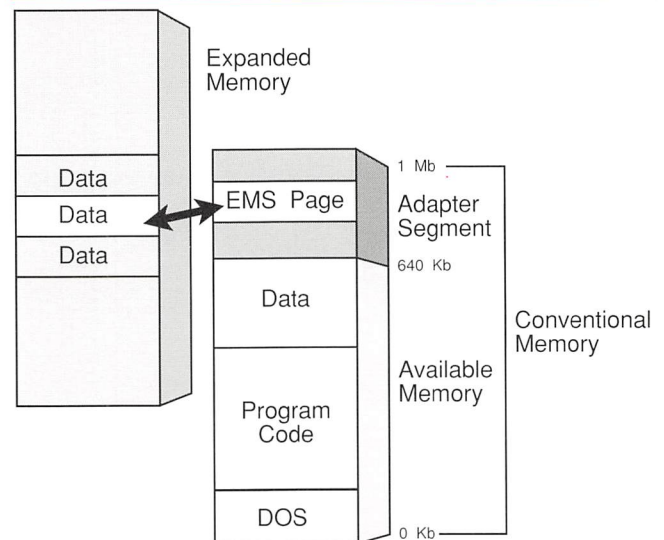
640-Kb capacity of DOS-based computers became the "640-Kb barrier." New generations of applications offering powerful new features could barely shoehorn themselves into 640 Kb of memory. Nevertheless, the need to maintain compatibility with the installed base of DOS computers dictated working within the confines of conventional memory, and the 640-Kb barrier became an albatross around the necks of programmers and PC users alike.

## Expanded memory (EMS)

Because applications so desperately needed more than 640 Kb of memory, Lotus, Intel, and Microsoft jointly developed a way to allow an 8088-based computer to access more memory than it was designed to use. The method they came up with is called the *Lotus-Intel-Microsoft Expanded Memory Specification*, or LIM EMS for short. Here's what it involves: First, you install in your computer an expanded memory add-in board. Additionally, you install a file on your boot disk called an expanded memory manager, which, as its name implies, manages the way data is moved into and out of the memory on the expanded memory board.

Once you've installed both the expanded memory board and the expanded memory manager, DOS can use the expanded memory by making requests to the expanded memory manager. As Figure B illustrates, the expanded memory manager creates a "window" in an unused section (usually about 64 Kb) of the adapter segment through which your application can view pages of data stored on the expanded memory board. This effect is similar to using your screen as a window to view sections of a large spreadsheet or word processor document.

**Figure B**



*The EMS memory manager swaps pages of information from expanded memory in and out of conventional memory.*

When DOS asks the expanded memory manager to store some data in expanded memory, the expanded memory manager copies that data into the designated section of the adapter segment, and then copies it out to the expanded memory.

Similarly, when DOS asks the expanded memory manager to retrieve some data that's currently stored in expanded memory, the expanded memory manager will find the desired data, then move that data into the designated section of the adapter segment where DOS can access it.

Because expanded memory is an add-on patch instead of an integral part of the PC's architecture, it has some significant limitations. For instance, under the widely accepted LIM EMS version 3.2 specification, applications can store particular pieces of data in expanded memory, but all of the application's program code must remain in conventional memory. This allows you to use an LIM EMS-compatible application to create and manipulate larger spreadsheets or database files than would be possible using conventional memory alone, but the 640-Kb barrier still limits the size of the application's program code. Additionally, because only parts of the application's data can be moved into expanded memory, you can still run out of conventional memory before you fill up your expanded memory.

A more recent modification to the LIM EMS standard (version 4.0) allows newer expanded memory cards to move program code as well as data into expanded memory. Although this allows expanded memory to accommodate very large application programs, applications that use this scheme typically run slowly, since the expanded memory manager must move page-sized blocks of data back and forth between conventional memory and expanded memory.

Because LIM EMS has been around for years and was the first widely accepted method for breaking the 640-Kb barrier, it is widely supported by a large number of applications. In fact, a few applications, such as Xerox's Ventura Publisher Profession Extension, won't even run on your system unless you've installed expanded memory.

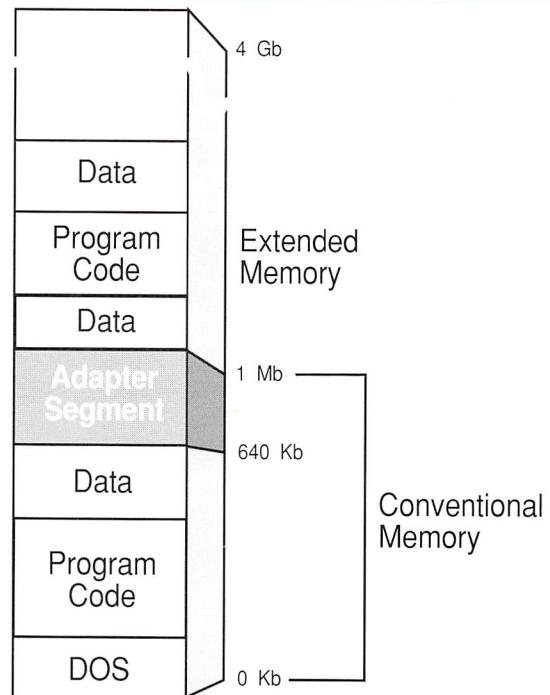
## Extended memory (XMS)

In addition to expanded memory (EMS), computers based on the 286, 386, and 486 processors can use *extended memory* (XMS). Installing extended memory typically involves installing modules of memory chips into empty slots on the computer's motherboard, then installing an extended memory manager that tells DOS how to use the extended memory.

As Figure C illustrates, extended memory is simply the memory that lies above the 1Mb line in a 286-, 386-, or 486-based system. In other words, extended memory

is a logical extension of the processor's total address space; the 286, 386, and 486 processors can address extended memory just as easily as they address conventional memory. (Incidentally, the 286 processor can address up to a total of 16 Mb of RAM, while the 386 and 486 processors can address up to 4 gigabytes, or 4000 Mb, of RAM.)

**Figure C**



*Extended memory is simply the memory that lies above the 1 Mb line in a 286-, 386-, or 486-based computer.*

Unfortunately, the current versions of DOS—and therefore DOS-based applications—cannot take full advantage of your computer's extended memory. Even if your machine has several megabytes of extended memory available, nearly all DOS applications will be able to address only the computer's first 640 Kb, just like it would on an 8088-based machine. As a result, most 286- and 386-based systems running today are simply fast 8088 machines, and aren't exercising their advanced capabilities.

The only way DOS developers can directly access extended memory is to incorporate a DOS extender directly into their application programs. Lotus 1-2-3 Release 3 is one example of a DOS application that uses a DOS extender to access extended memory.

Incidentally, you've probably heard a lot of talk about the new version of Microsoft Windows—version 3.0. One of the compelling reasons to run version 3.0 of Microsoft Windows on top of DOS is that applications written to run under Microsoft Windows aren't limited by DOS' 640-Kb barrier. Instead, Windows applications

can directly address all the extended memory installed in your machine. This allows developers to create new, feature-laden applications without worrying about the confines of DOS' 640-Kb barrier, and it allows users to solve their memory problems by installing as much extended memory as they need.

Hopefully, Microsoft will release a new version of DOS sometime in the near future that will provide the same support for extended memory as that offered by the latest version of Microsoft Windows. That way, DOS developers and users would be able to access the full complement of extended memory installed in their machines.

## Real mode vs. protected mode

So far in this article, we've intentionally avoided using a couple of important terms in our discussion of the 286, 386, and 486 processors: *real mode* and *protected mode*. However, in order for you to gain a better understanding of how 286, 386, and 486 processors access memory, we need to define these two operating modes.

Real mode and protected mode are the two possible modes of operation available to 286, 386, and 486 processors. You ought to think of real mode as 8088-compatibility mode. Whenever you run a typical DOS application on a 286-, 386-, and 486-based machine, the processor runs in real mode. Unfortunately, running a processor in real mode precludes it from taking advantage of the processor's advanced capabilities—including its ability to access extended memory.

In order to access the advanced capabilities of 286-, 386-, and 486-based systems, you must run the processor in protected mode. While the 286 processor is running in protected mode, it can directly access as much as 16 Mb of extended memory. Running the 386 and 486 processors in protected mode not only allows them to access up to 4 gigabytes of extended memory, it also allows these processors to set up and run multiple 8088 sessions simultaneously in extended memory—a popular concept known as *multitasking*.

Until Microsoft provides a version of DOS that allows DOS applications to run in the processor's protected mode, the only way for developers to access extended memory directly is to use the method we mentioned earlier—to incorporate a DOS extender directly into their application programs. A DOS extender will allow the developer to throw the processor into protected mode, and access the computer's extended memory.

## What should you do?

After reading about the various types of memory and operating modes, you're probably wondering "What does all of this mean for me?" What all of this boils down to is as follows: When you need to install additional memory

in your machine, you'll want to make sure you purchase the right kind of memory. Although you might be tempted to let your current suite of applications dictate the type of memory you should buy, you should instead base your decision upon the kind of processor in your machine.

If you are working with an 8086- or 8088-based system and you need more memory, your only option is to purchase and install expanded memory. As we discussed earlier, this simply involves installing an add-in memory board, copying an expanded memory manager file onto your hard disk, and modifying your CONFIG.SYS file.

If you use a 286- or 386-based system, however, you can either install expanded memory or extended memory. Regardless of the applications you use, we strongly recommend that you install extended memory—not expanded memory. Here's why: In the months and years ahead, you'll see most major operating systems and applications shift from support for expanded memory to support for extended memory. In fact, as we mentioned earlier, the current version of Microsoft Windows takes full advantage of extended memory, and a few current DOS applications, such as Lotus 1-2-3 Release 3, require it.

In the meantime, if you're using a 286- or 386-based machine, and you need to provide expanded memory to your applications, you should install extended memory, and then install a memory management utility that allows your applications to use extended memory as if it were expanded memory. Some packages that provide this function are Microsoft Windows, QEMM from QuarterDeck, and 386 to the Max from Qualitas. You need to be aware, however, that while a 386- or 486-based machine can emulate in extended memory both versions 3.2 and 4.0 of LIM EMS, a 286-based machine can emulate only version 3.2 of LIM EMS. To emulate LIM EMS 4.0 on a 286-based machine, you have to install an expanded memory add-in board, which contains some extra circuitry present in the 386 and 486 processors, but not in the 286.

## Conclusion

Because the IBM PC's original microprocessor, the 8088, is limited to a 1 Mb of address space, DOS is confined to working in 640 Kb of memory. If you want to provide more memory to your applications, you need to understand the difference between conventional, expanded, and extended memory, which we've covered in this article.

If you have an 8088- or 8086-based machine you'll need to buy expanded memory. However, if you have a 286-, 386-, or 486-based machine, you'll want to buy extended memory—not expanded memory. Even if your applications require expanded memory, you're better off buying extended memory, and then installing a memory management utility that emulates expanded memory in extended memory. ■

# Issue a batch file alert by ringing your computer's bell

If you're like most DOS users, you send a variety of messages and prompts to the screen from within your batch files. Some of the messages you send are informational in nature (such as *Copying files, please wait...*), while others are much more important (such as *You've made a critical error—please reboot your machine*).

One way to call special attention to important batch file messages—or to anything important that's going on in the batch file—is to ring the computer's bell. To ring the bell, simply use an ECHO command to send a [Ctrl]G to the computer. Just type *echo*, followed by a space, and then press [Ctrl]g. When you do this, your current line will look like this:

```
echo ^G_
```

Notice that DOS uses the characters ^G to represent the key combination [Ctrl]G. As soon as you press ↵ to issue this command, DOS will ring the computer's bell. (Actually, your computer doesn't have a bell—it has a small speaker that "beeps." It's much easier, however, to say "ring the bell" than "send a beep to the computer's speaker.")

By the way, if you use the TYPE command to display a batch file containing the command *echo ^G*, you won't see the characters ^G. Don't let this trouble you. When you use your text editor to edit the file, the characters ^G will appear in their proper places.

## An example

As an example of how you can effectively use a bell ring, consider the batch file shown in Figure A, which creates a backup copy of the files in the directory C:\123\DATA.

**Figure A**

```
@echo off
echo Backing up to drive E...
xcopy c:\123\data e:\123\data
echo Backup finished
echo ^G
```

*This batch file rings the bell after it backs up the data files.*

As you probably know, the first command in this batch file

```
@echo off
```

tells DOS to suppress the display of the batch file's commands as it executes them. The second command

```
echo Backing up to drive E...
```

tells DOS to display the message *Backing up to drive E...* The batch file's third command

```
xcopy c:\123\data e:\123\data
```

tells DOS to copy all the files in the directory C:\123\DATA to the directory E:\123\DATA. Of course, if you've stored a lot of data in the directory C:\123\DATA, you might have enough time to take a short nap while you wait for the XCOPY command to complete its job. When it's finally finished, however, DOS will execute the command

```
echo Backup finished
```

which tells you that the copy procedure is complete. Finally, DOS will execute the last command in the batch file

```
echo ^G
```

which rings the bell to notify you that the backup process is over. (If you're lucky, the bell will ring loudly enough to wake you up and remind you that you can finally get back to work.)

## Other uses for the bell

In addition to using the bell for signalling the completion of a task, you might want to use it to signal anticipated errors in a batch file. For example, if your batch file prompts you to press a particular key from a list of valid choices, and you press an invalid key, the batch file could ring the bell and display the message

```
Sorry, you pressed an invalid key
```

Another situation where the bell comes in handy is when you need to instruct the user to do something important, like change disks or reboot the machine. Don't overdo it, though—if your batch file rings the bell too often, the bell will be less effective in commanding special attention. Even worse, too many bell rings can annoy the user and make him or her wish the batch file would simply be quiet.

## Conclusion

Ringing the computer's bell is an effective way to command attention. In this article, we've shown you how to use the ECHO command to ring the bell, and we've discussed some effective ways to employ this technique in a batch file. ■



## Guidelines for managing your fixed disk

**I** imagine this scenario: Federal Express is coming in 15 minutes and you still have to print the 20-page proposal you've been working on for three days. You make the last change and save the final version, but instead of the comforting sound of your fixed disk working you see the dreaded message telling you that the disk is full. There's no time for thoughtful analysis, so you quickly delete several files to make room. A couple of days later you realize that you wiped out a file that contained most of your research.

That's if you're lucky. Some application programs don't let you delete files without exiting to DOS, and if you can't find a disk with enough room to store the file, you'll lose all the work from your latest session and might well miss the deadline.

And the computer was supposed to make life easier?

### There's no free lunch

Like your paper files, the usefulness of your computer files depends primarily on how well you organize your filing system and how faithfully you keep it sorted and pruned. If you let your paper files pile up on your desk or in one big drawer, it takes longer and longer to find a particular file; if you don't break down your files into smaller categories and put meaningful labels on the folders, it still can take too long to find something.

And even if you're diligent about filing, the paper periodically overflows your file drawers and you've got to take the time to move the less-important files to archive storage boxes and toss out the ones you don't need (or can't remember why you kept).

Computer files are much the same. If you let all your files pile up in the root directory, pretty soon you can't find anything. Even if your word processing or spreadsheet program is in its own directory, if you keep storing all the data files in the same directory, sooner or later you won't be able to remember the significance of those filenames you once knew you'd never forget.

And no matter how well you set up your computer filing system, periodically you've got to purge the files you don't need—after copying to archive disks those you should keep another year or two, of course. If you don't, your fixed disk will fill up at the worst possible moment.

It takes time to set up and maintain your computer filing system, but the investment pays off by making your system more efficient. Your time and the information stored in your computer are probably your most valuable assets. The longer it takes you to find a file, the less time you have for productive work; if you can't find a file, you may as well not have it.

### The bare root policy

The first guideline for managing your fixed disk effectively is to keep the root directory as free of files as possible. It's much easier to keep track of your files if the root directory contains nothing but subdirectories for application programs and the three files that DOS requires to be there—COMMAND.COM, AUTOEXEC.BAT, and CONFIG.SYS.

There can be a few exceptions to this rule. For example, some programs (such as QDOS and Mace Utilities) create a file in the root directory that contains a description of the disk's file structure. This file lets the program reconstruct the disk should the file structure become damaged or lost. The value of this insurance is worth having the extra file in the root.

Put all the DOS files in a subdirectory of the root directory named \DOS (version 4 does this for you when you install it). If your DOS files are in the root directory now, create a subdirectory named \DOS, copy the DOS files to the subdirectory, and delete them from the root directory. If you have one or more devices that require a special program called a *device driver*—such as a scanner or CD-ROM drive—either put these files (their extensions are usually SYS) in the DOS directory, too, or create their own subdirectory named \DRIVERS.

### Organize your subdirectories

Create a separate subdirectory in the root directory for each application program. Most programs offer to create a subdirectory for you when you install them. Unless you already have a directory with the name your program suggests, accept its offer. These subdirectories usually go in the root directory, so you'll wind up with subdirectories named something like \WP for word processing and \123 for your spreadsheet.

After you have created subdirectories for DOS and all your application programs, make sure that your AUTOEXEC.BAT file includes a PATH command that names each of these subdirectories. This lets you use any DOS command or application program without changing the current directory.

For instance, if the only subdirectories on your fixed disk are named \DOS, \WP, and \123, you'll want to include the command

```
path=c:\dos;c:\wp;c:\123
```

in your AUTOEXEC.BAT file.

## Don't forget your data files

Unless an application program requires that its data files—word processor documents, for example, or spreadsheets—be stored in the same directory as the program itself, it usually isn't a good idea to store the data files with the program. An application program might be made up of a dozen or more files; it's easier to find your data files (and sometimes easier to upgrade to a new version of the application) if you store the data files separately from the program.

Unless you create only a few files, you should probably create separate subdirectories for different types of work. If you serve several different clients, create a separate subdirectory for each client (GM, KODAK, INTEL, etc). If all your work is for yourself or a single client, you could create subdirectories for each type of work: under \WP you could have LETTERS, PROPOSLS, and REPORTS; under \123 you could have BUDGETS, SALES, INVENTORY, and TAXES.

As your data files proliferate, you'll discover more breakdowns that let you keep track of where things are. In general, you should probably start thinking about breaking up a subdirectory when it grows beyond 25 or 30 files. As you create more subdirectories, add them to the application directory rather than one of the data subdirectories; try to keep the total depth of your filing structure to three or four levels. These kind of directory names—\WP\LETTERS or \123\SALES\WEST—keep your path names reasonably short, but still allow you to break up your data files into meaningful groups.

## Purge your files regularly

Don't wait until you run out of disk space before deleting unneeded files. Once a week or so, depending on how much you use your system, go through all your data file subdirectories, copying to archive disks the files for which you have no immediate need but should save for permanent records, and deleting the files you don't need at all (especially files with BAK extensions). If you don't do this regularly, someday you'll find yourself doing it hastily, increasing the chance for mistakes; in some cases, you may even lose work because you don't have enough disk space to save the revised version of a file.

## Back up, back up, back up

And last, but most important, back up your data files regularly. System failures—both hardware and software—do happen; although they're blessedly infrequent, they usually happen when you can least afford the loss. More frequently, however, we hurt ourselves, mistakenly deleting the wrong files or entire directories; after such pilot error, you can escape most of the anguish by reaching for your most recent backup disk and recovering most or all of the lost files.

You have the master disks of your program files so you don't need to back them up, but you must protect your data files. If you have a backup program, such as Fastback Plus or Norton Backup, follow its instructions for identifying the directories and files you want to back up. If you don't have a backup program, create your own backup batch file using either the DOS BACKUP or XCOPY command to back up any data files that have changed since they were last backed up.

No matter how you set up your backup procedure, make it a habit. Don't cheat to save a few minutes; inevitably, the time you most need the backup copy of a file will be the day after you were in too much of a hurry to take a few minutes to back up.

## It's worth the time

Few things are as frustrating as knowing that a file you want is somewhere on your fixed disk, but not being able to pick the file out of the clutter. If you're setting up your system for the first time, following these guidelines for setting up and maintaining your filing system will keep you from getting lost in a welter of files.

If you've been using your system for a while and your file system isn't organized this way, take a few hours to rearrange things. It's one of the cheapest investments you can make in improving the efficiency of your computer and your own productivity. ■

## OTHER COBB GROUP JOURNALS

### DOS applications

*FOR QUATTRO*

*Inside WordPerfect®*

*Paradox User's Journal*

*Symphony User's Journal*

*The Workshop (Microsoft Works)*

*Word for Word (Microsoft Word 5.0)*

*1-2-3 User's Journal*

*Inside 1-2-3 Release 3*

### Windows applications

*Inside Microsoft Windows*

*Inside Word for Windows (Microsoft Word for Windows)*

*The Expert (Microsoft Excel)*

### Languages

*Inside Microsoft BASIC*

*Inside Microsoft C*

*Inside Turbo Pascal*

*Inside Turbo C*

To order a subscription to any of these journals, call The Cobb Group toll free at 1-800-223-8720.

## Formatting bootable disks from drives D, E, and F

**Q** I have a problem that I hope you can solve. I have a large hard drive that is partitioned into four separate volumes: C, D, E, and F. I've stored all of my DOS 3.3 files in the directory C:\DOS, and I've included that directory on the path.

Here's my problem: When I make drive D, E, or F the current drive, then try to format a bootable disk in drive A using the command

```
D>format a: /s
```

DOS doesn't display the expected message

Insert new diskette for drive A:  
and strike ENTER when ready

Instead, I receive the message

Insert DOS disk in drive A:  
and strike ENTER when ready

However, if I type `c:` and press `↵` to make drive C the current drive, then issue the same `FORMAT` command from a `C>` prompt, DOS formats the bootable disk without a hitch.

## Backing up with XCOPY, revisited

**Q** I was happy to see Delores Williams' letter "Using XCOPY Instead of BACKUP" in the August issue of *Inside DOS*. I have been looking for a way to make uncompressed copies of all the files on my hard disk, and her form of the `XCOPY` command did the trick; it overcomes the limitations imposed by `BACKUP` and `COPY`.

After experimenting with her technique, though, I thought it might be worthwhile to add a few notes of interest. First of all, the `XCOPY` command requires blank, formatted disks. Be sure to have plenty on hand before you begin.

Additionally, since you'll want to make a copy of every file on your hard disk the first time you back up, you'll need to "initialize" the archive bits for all your files so that `XCOPY` won't skip over any of them. In other words, you need a way to turn on the archive bits for all your files. Fortunately, you can do this easily by issuing the command

```
attrib +a c:\*.* /s
```

Once you've issued this command, you can begin the process of backing up every file on the hard disk using the command

Why can I successfully format bootable disks only when drive C is the current drive?

Paul Jacobson  
Dallas, Texas

**A** As you've discovered, Mr. Jacobson, including the directory C:\DOS in your path allows DOS to find and run the `FORMAT` command successfully. In order to create a bootable disk, however, `FORMAT` needs to find the three system files `COMMAND.COM`, `IBMBIO.COM`, and `IBMDOS.COM`, which are all installed in the root directory of drive C. (The files `IBMBIO.COM` and `IBMDOS.COM` are hidden, so you won't see them in a directory listing.)

Unfortunately, `FORMAT` doesn't use the path to look for these system files. Instead, it looks only in the root directory of the current drive. Because drives D, E, and F don't have these files in their root directories, `FORMAT` asks you to insert a disk containing those files (which can be any bootable disk) in drive A.

Obviously, we'd like the `FORMAT` command in future versions of DOS to read the system environment variable `COMSPEC` to find out where the computer's system files are stored. That way, you could use the `FORMAT` command to format a bootable disk from any current drive—not just the drive from which you boot.

```
xcopy c:\*.* a: /s /m /v
```

Notice that I've included the parameter `/v` in this command, which tells DOS to verify files as it goes along—a nice safeguard that I recommend highly for a backup procedure.

When DOS fills up the first disk with files, it will present the message

**Insufficient disk space**

and return you to the DOS prompt. At that point, you can replace the disk in drive A with a new one, and press `[F3]` to reissue the `XCOPY` command. As Ms. Williams explained in her letter, DOS will pick up where it left off, and continue copying additional files from drive C to drive A.

As with all floppy-disk backup procedures, this backup technique requires you to spend a lot of time waiting for and administering the backup procedure. If you're patient, however, you'll end up with a complete copy of every unhidden file on your hard disk.

One more thing: If you want to exclude only a file or two from your "universal" backup, you can do so easily.

Simply follow the ATTRIB command listed above with another ATTRIB command that takes the form

```
attrib -a filename
```

where *filename* is the full path name of the file you want to exclude from the backup.

For example, suppose you want to make backup copies of all the files on drive C, except for the file C:\123\BUDGET.WK1. To do this, you would use the following three commands:

```
C>attrib +a c:\*.* /s
C>attrib -a c:\123\budget.wk1
C>xcopy c:\*.* a: /s /m /v
```

The first command turns on the archive bit of every file on drive C. The second command turns off the archive bit of the file C:\123\BUDGET.WK1. Finally, the

third command tells DOS to copy to drive A all the files on drive C whose archive bit is turned on.

Greg Laws  
Winfield, Kansas

Mr. Laws' insights into this technique are very important. As he points out, failing to use the ATTRIB command before issuing the XCOPY command will result in incomplete backups. By following Mr. Laws' guidelines, however, you can ensure that DOS will back up all the files you specify.

Incidentally, Mr. Laws' letter demonstrates only one of the many powerful uses for the ATTRIB command. In future issues of Inside DOS, we'll show you some additional uses for ATTRIB, including how to change a file's status to read-only, and how to copy all files except those you specify into another directory. ■

## ADVANCED COMMAND TECHNIQUE

# Using the FOR command to issue several commands at once

Have you ever wanted to execute a command more than once in a batch file, and were forced to type in those commands one after the other, like this:

```
xcopy c:\sys\*.* a: /s
xcopy c:\data\*.* a: /s
xcopy c:\batch\*.* a: /s
```

If so, you can take advantage of the FOR command. FOR allows you to use a single command to issue several commands at once.

Although FOR is a little more complex than most DOS commands, you can really save yourself some time, and improve the efficiency of your batch files, once you understand how to put this command to work for you.

The FOR command takes the form

```
for %%p in (set) do command
```

You must include the words *in* and *do* in the FOR command—they are not arguments. The argument %%p is a replaceable parameter. As you'll see in a moment, DOS assigns to %%p, in turn, each value in *set*.

The *set* argument specifies the list of possible values DOS will assign to %%p. You must separate the items in *set* with spaces and place parentheses around the entire list, like this: (1 2 3 4). If you want to use filenames as

the elements of *set*, you can use DOS wildcards to specify those names, as in (c:\*.bat).

The argument *command* is the DOS command you want to carry out more than once—it can be any DOS command other than FOR. Typically, *command* will include at least one replaceable parameter—either the FOR command's replaceable parameter, %%p, or a batch command parameter, like %1. (For a detailed discussion of batch command parameters, see the article in the July issue entitled "Creating Batch Commands that Operate on Variable Data.")

Let's take a moment to consider what happens when you issue a FOR command. First, DOS assigns to the variable %%p the first value in *set*, and carries out *command*. Next, DOS assigns to the variable %%p the second value in *set*, and once again carries out *command*. DOS continues this process until it has assigned to %%p every value in *set*.

If you don't yet understand how the FOR command works, don't worry. Like most DOS commands, FOR is much easier to understand once you've seen some examples of its use.

One more thing before we get into the examples: You don't have to use %%p as the FOR command's replaceable parameter—you can substitute for the letter *p* any character other than one of DOS' redirection symbols (<, >, or |). For instance, you can use %f or %1 in place of %%p.

## An example

Let's demonstrate the FOR command by using it to do something very simple. For instance, suppose you want to include the following three ECHO commands in a batch file:

```
echo land
echo air
echo sea
```

Instead of entering all three of these commands, you can use the following FOR command:

```
for %p in (land air sea) do echo %p
```

As you can see, the *command* argument of the FOR command is

```
echo %p
```

Consequently, the FOR command will repeatedly issue this ECHO command, substituting each of the words you've included in *set* for %p. To test this command, create a batch file named FORTEST1.BAT by typing

```
C>copy con fortest1.bat
for %p in (land air sea) do echo %p
^Z
1 File(s) Copied
```

Now, run this batch file by issuing the command

```
C>fortest1
```

Because you didn't include an *echo off* command in your batch file, DOS will display the batch file's commands as it executes them, starting with the FOR command

```
C>for %p in (land air sea) do echo %p
```

At this point, DOS repeatedly carries out the ECHO command for each value in *set*, substituting each value (land, air, and sea) for the replaceable parameter %p in the ECHO command:

```
C>echo land
land
```

```
C>echo air
air
```

```
C>echo sea
sea
```

You might have noticed in this example that DOS drops one of the % signs off of the parameter %p when it displays the FOR command on the screen. Don't let this trouble you—DOS leaves one % sign in front of the letter *p* so you can still identify the item as a replaceable parameter.

## Including batch command parameters

As we mentioned earlier, you can include batch command parameters, such as %1, in your FOR commands. However, because you can have both %1 and %p in the same command, make sure you remember the difference between these two types of parameters. The %p parameter refers to one of the values in the *set* argument of the FOR command, whereas the %1 parameter refers to the first parameter you type on the command line, the %2 parameter refers to the second parameter you type on the command line, and so forth. Let's clarify this concept with a simple example.

Suppose you want to create a batch file named FORTEST2.BAT that displays the three words you supply as parameters to the batch command. One way you could do this is by including the following three commands in the file:

```
echo %1
echo %2
echo %3
```

However, you can instead use a single FOR command that performs the same task:

```
for %p in (%1 %2 %3) do echo %p
```

To create a batch file that contains this command, just type

```
C>copy con fortest2.bat
for %p in (%1 %2 %3) do echo %p
^Z
1 File(s) copied
```

The FOR command in this batch file tells DOS to carry out the command *echo %p* three times, substituting for %p, in turn, each of the three words you type on the command line.

To test the FOR command, simply type

```
C>fortest2 quick brown fox
```

Immediately, DOS displays the FOR command

```
C>for %p in (quick brown fox) do echo %p
```

As you can see, DOS has already substituted the words *quick*, *brown*, and *fox* for the parameters %1, %2, and %3 in the FOR command's *set* argument.

After DOS displays the FOR command, it carries out the command *echo %p* three times, substituting for %p each of the three words you supplied on the command line:

```
C>echo quick
quick
```

```
C>echo brown
brown
```

```
C>echo fox
fox
```

## Specifying filenames in the *set* argument

All the sample FOR commands we've shown so far use literal words (like *quick*, *brown*, and *fox*) as their *set* argument. Most of the time, however, you'll want to specify a list of filenames for *set*. To do this, you can either type out the name of each file individually, or you can use the wildcard characters \* and ? to specify the appropriate file names.

For example, suppose you want to copy to drive A every file in the current directory whose extension is TXT. (Of course, you could easily do this with the COPY or XCOPY command, but we'll use this scenario for the sake of example.) The FOR command that will do this is

```
C>for %p in (*.txt) do copy %p a:
```

This command tells DOS repeatedly to issue the command

```
copy %p a:
```

substituting, in turn, the name of every file in the current directory whose extension is TXT.

Unfortunately, the FOR command doesn't let you use wildcards to specify a list of directory names. For instance, if you want to remove all the subdirectories from the current directory, you might be tempted to use the command

```
C>for %p in (*.*) do rd %p
```

As soon as you issue this command, however, DOS will display the message

**Syntax error**

and will return you to the DOS prompt, indicating that directory name wildcards aren't allowed in *set*.

## Using batch commands in the *command* argument

As we mentioned earlier, the *command* argument in a FOR command can be any DOS command. If you specify a batch command as your *command* argument, DOS won't carry out *command* for every value in *set*. Instead, it will carry out *command* for the first value in *set*, and then will immediately return to the DOS prompt.

To carry out a batch command for every value in *set*, you need to perform an extra step. If you're using DOS 3.3 or later, just precede the name of the batch command with the word *call*. For instance, if you want to run the batch files

```
TEST1.BAT
TEST2.BAT
TEST3.BAT
```

one after the other, you can use the command

```
C>for %p in (1 2 3) do call test%p
```

If you're using version 3.2 or earlier, however, you'll need to precede the batch command with *command /c* instead of *call*, like this:

```
C>for %p in (1 2 3) do command /c test%p
```

If you fail to precede the command *test%p* with either *call* or *command /c*, DOS will return to the DOS prompt immediately after it executes the batch file TEST1.BAT.

## Conclusion

In this article, we've introduced you to the FOR command, which lets you issue several similar commands at once. Now that you're familiar with the basics of this command, read the article on the back page entitled "Tightening up Batch Files Using the FOR Command," which shows you a practical application for this command. Of course, we'll present some additional tips and techniques that use the FOR command in future issues of *Inside DOS*. ■

### INSIDE DOS BACK ISSUES

Back issues of *Inside DOS* are a handy resource. If your *Inside DOS* library is incomplete, you should invest in back issues. These issues are available for \$4 each. To order back issues, call The Cobb Group toll free at 1-800-223-8720. A free index is available upon request.



Postmaster:  
Monthly Technical Journal  
Please Rush!  
Second Class Postage

Please include account number from label with any correspondence.

## BATCH FILE TIP

# Tightening up batch files using the FOR command

*In the article "Using the FOR Command to Issue Several Commands at Once" on page 9, we introduce you to DOS' FOR command. This article shows you one practical application for the FOR command.*

If you use the IF command in your batch routines to check a particular condition, you'll probably end up with some clumsy, repetitive program code. For instance, if you write a batch routine that requires you to check the value of the system variable ERRORLEVEL, you'll probably end up with some code like that shown in Figure A.

Figure A

```
keypress
if errorlevel 3 goto err3
if errorlevel 2 goto err2
if errorlevel 1 goto err1
:ERR3
    err3 commands
:ERR2
    err2 commands
:ERR1
    err1 commands
```

*This batch code includes three consecutive IF commands.*

As you can see, Figure A includes three consecutive IF commands that check the value of ERRORLEVEL, then tell DOS to branch to the appropriate section of the batch file (ERR3, ERR2, or ERR1).

Whenever you see consecutive batch commands that are nearly identical to one another, like the three IF commands in the above routine, you can use the technique we explain in the article on page 9 to condense those commands into a single FOR command. In this

case, you can change the sample block of code to that shown in Figure B.

Figure B

```
keypress
for %%p in (1 2 3) do if errorlevel %%p goto err%%p
:ERR3
    err3 commands
:ERR2
    err2 commands
:ERR1
    err1 commands
```

*We've tightened up the code in Figure A by replacing the three IF commands with a single FOR command.*

Notice that the latter section of code is shorter and, therefore, requires less disk space than the former section. However, the series of IF statements in the former section are probably easier to write and to maintain on an ongoing basis. Consequently, you should use the former approach when ease of writing and maintenance is important, but you should take the latter approach when you want the batch file to contain as few commands as possible.

By the way, you might have noticed that we reversed the order of the numbers 3, 2, and 1 in our FOR command. We had to do this in order to allow the FOR command to successfully branch to the appropriate section of the batch file. Whenever you use the FOR command to perform this type of comparison test with the variable ERRORLEVEL, make sure you place the values in the FOR command's *set* argument in ascending order. Otherwise, the FOR command won't successfully branch to the appropriate section of the batch file. ■